

Accelerator Description Exchange Format ADXF 2.0

N. Malitsky and R.Talman

Revision: January 20, 2005
The first draft: January 11, 2005

Outline

- ❑ ADXF Objectives and Use Cases
- ❑ Dictionary of Accelerator Model Concepts
- ❑ UML Model
- ❑ UML-to-XML Schema Mapping
- ❑ XML Schema
- ❑ Examples
- ❑ Appendix A: ADXF vs MAD
- ❑ Appendix B: ADXF vs SXF
- ❑ Appendix C: ADXF Accelerator vs ROOT Detector

Accelerator Description Exchange Format (ADXF)

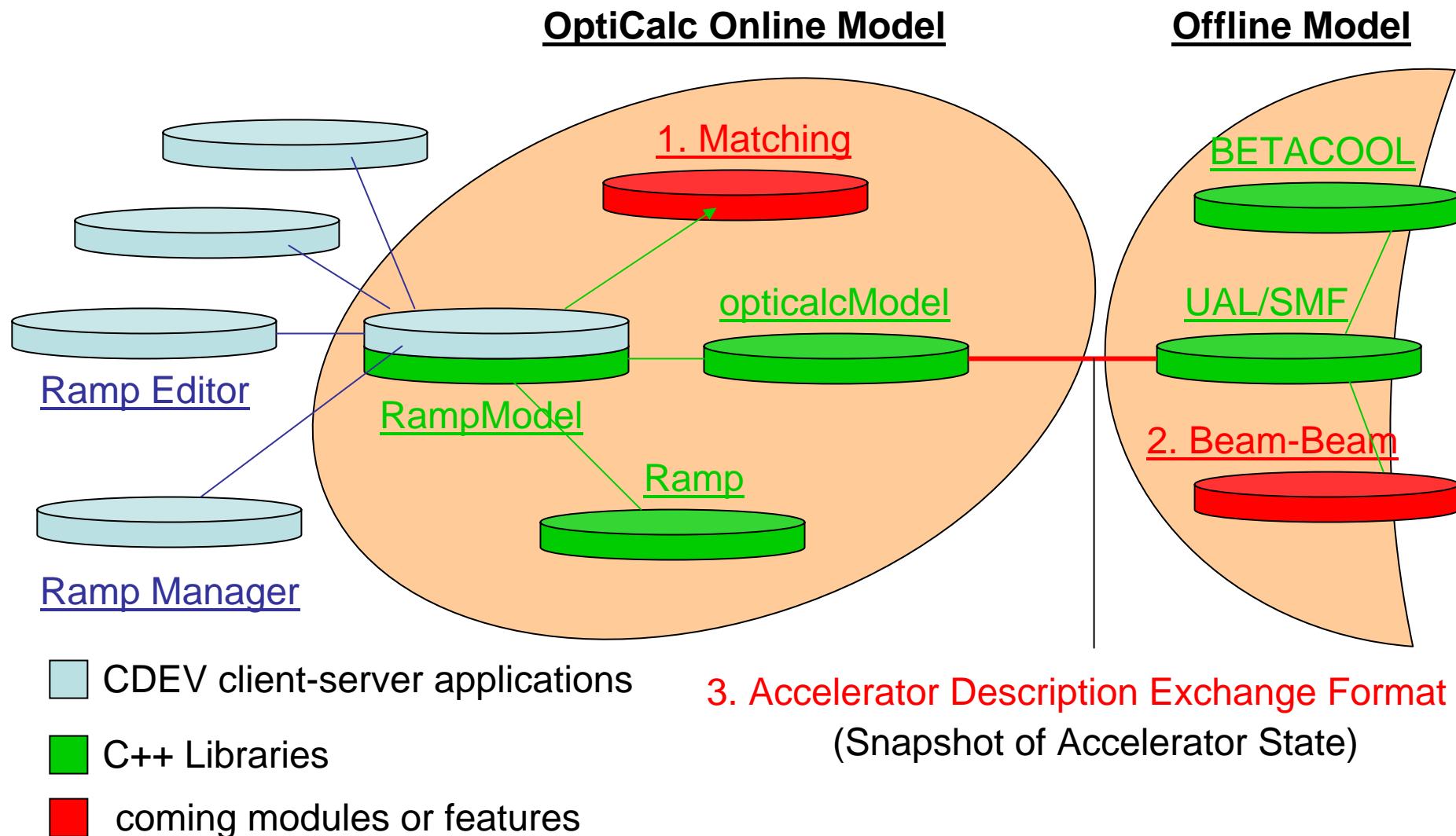
ADXF 1.0 (1998)

- ❑ **Response** to the Iselin-Keil-Talman Accelerator Description Standard (ADS) request
- ❑ **Based** on the XML technology
- ❑ **Integrated** the MAD sequence and UAL element attribute sets

ADXF 2.0 New Objectives and Features (now)

- ❑ **Inspired** by E.Forest's "fibre bundles" and ROOT Detector geometry description.
- ❑ **Adds** a new concept, "installed" element.
- ❑ **Addresses** several issues:
 - ❑ sharing of common "real" elements by several sectors;
 - ❑ supporting of design and operational accelerator descriptions;
 - ❑ connecting of accelerator and detector models;
 - ❑ TBD

Joining the RHIC Online and Offline Models

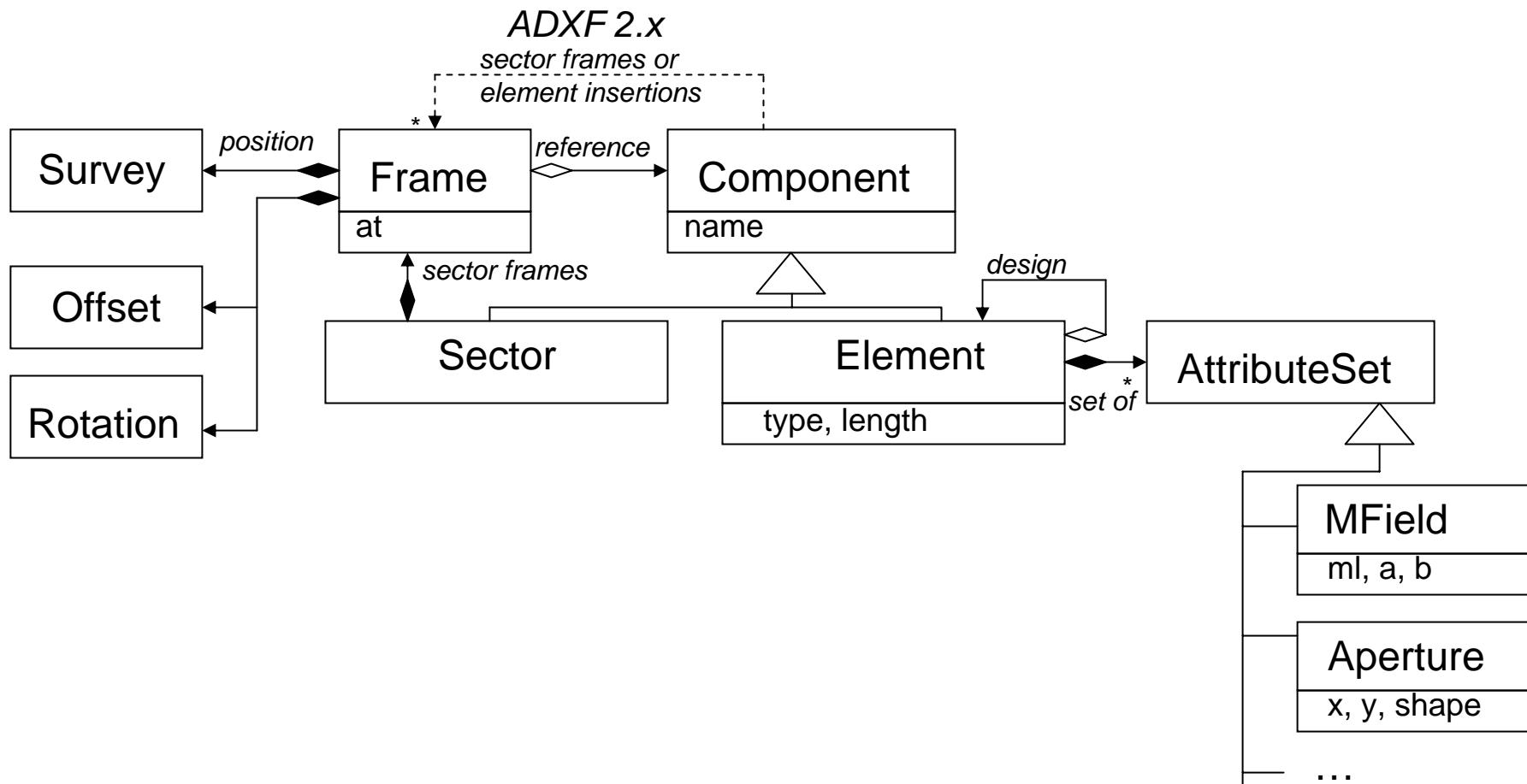


Dictionary of Accelerator Model Concepts

- **Accelerator** is any sector selected by user.
- **Sector** is a named sequence of frames with installed accelerator components.
- **Frame*** is a layout of an installed component. It contains a relative position (longitudinal position “at” or Survey coordinates), misalignments, and a reference to an associated component, sector or accelerator element.
- **Accelerator Element** is a leaf component in the accelerator tree organization. There are many different types of accelerator elements (e.g. sbend, quadrupole, etc.) But all of them have the same structure: name, length, and an open collection of attribute sets. Element may have a reference to the design element.
- **Element Attribute Set** is a container of attributes relevant to the single physical effect or feature (e.g. magnetic field, aperture, etc.)

*Nick Walker's term

UML Model



UML-to-XML Schema Mapping

| # | UML Modeling Concept | C++ Example | XML Schema Concept | XML Schema Example with strong typing |
|---|---|----------------------------|----------------------------------|---|
| 1 | class | Class Frame { ... }; | element complexType | <element name = "frame"> <complexType> ... </complexType> </element> |
| 2 | primitive types: integer, double, etc. | double at; | attribute | <attribute name = "at" type = "double" /> |
| 3 | Collection of primitives | list<double> b; | simpleType, list attribute | <simpleType name = "doubleList"> <list itemType="double" /> </simpleType> <attribute name="b" type="doubleList" /> |
| 3 | reference or pointer | Component* ref; | attribute | <attribute name = "ref" type = "string" /> |
| 4 | collection of objects: set, list, etc. | list<Frame> frames; | sequence | <sequence> <element name = "frame" minOccurs="0" maxOccurs="unbounded" /> </sequence> |

UML-to-XML Schema Mapping (continued)

| # | UML Modeling Concept | C++ Example | XML Schema Concept | XML Schema Example with strong typing |
|---|----------------------|---|---|---|
| 5 | inheritance | Class MField : public AttributeSet { ... }; | element complexType extension substitution group | <complexType name="attributeSetType"/> <element name="attributeSet" type="attributeSetType"/> <complexType name="mfieldType"> <complexContent> <extension base="attributeSetType"> ... </extension> </complexContent> <complexType> <element name="mfield" type="mfieldType" substitutionGroup="attributeSet"/> |

ADXF 2.0

Definition:

ADXF 2.0 constitutes of parameters, elements, and sectors.

XML Schema:

```
<element name = "adx">
<complexType>
    <sequence>
        <element ref="parameters" minOccurs="0" maxOccurs="1" />
        <element ref="elements" minOccurs="0" maxOccurs="1" />
        <element ref="sectors" minOccurs="0" maxOccurs="1" />
    </sequence>
</complexType>
<unique name="parameterName">
    <selector xpath=".//constants/constant | .//constants/array"/>
    <field xpath="@name"/>
</unique>
<unique name="componentName">
    <selector xpath=".//elements/element | .//sectors/sector"/>
    <field xpath="@name"/>
</unique>
</element>
<element name = "parameters">
...
</element>
<element name = "elements">
...
</element>
<element name = "sectors">
...
</element>
```

ADXF Constant

Definition:

A **Constant** is a named parameter whose value can be defined by expression.

XML Schema:

```
<element name="constant">
  <complexType>
    <attribute name="name"/>
    <attribute name="value" type="expressionType"/>
  </complexType>
</element>
<simpleType name="expressionType" >
  <list itemType="string" />
</simpleType>
```

Example:

```
<!-- constant c1 whose value is 4.5 -->
<constant name="c1" value="4.5" />

<!-- constant c2 whose value is defined by expression -->
<constant name="c2" value="1.5*c1" />

<!-- array of two multipole harmonics -->
<constant name = "b2" value = "0.0 0.05" />
```

ADX Sector

Definition:

Sector is a named sequence of frames containing installed accelerator components.

XML Schema:

```
<element name = "sector">
  <complexType>
    <attribute name="name" />
    <sequence>
      <element name="frame" minOccurs="0" maxOccurs="unbounded" />
    </sequence>
  </complexType>
</element>
```

Example:

```
<!-- sector s1 with one installed element q1 -->
<sector name=s1" >
  <frame at="1.5" ref="q1"/>
  ... other frames
</sector>

<!-- sector s2 and sector s3 sharing the same sector s1 (and all its installed elements) -->
<sector name="s2">
  <frame at="1.1" ref="s1" />
  .... other frames
</sector>
<sector name="s3">
  <frame at="1.2" ref = "s1" />
</sector>
```

ADXF Frame

Definition:

Frame is a layout of an installed component (which represents one or more elements). It contains a relative position (longitudinal position “at” or six coordinates), misalignments, and a reference to an accelerator component (sector or accelerator element).

XML Schema:

```
<element name = "frame">
  <complexType>
    <attribute name="at" />
    <attribute name="ref" />
    <sequence>
      <element name="survey" minOccurs="0" maxOccurs="1" />
      <element name="offset" minOccurs="0" maxOccurs="1" />
      <element name="rotation" minOccurs="0" maxOccurs="1" />
    </sequence>
  </complexType>
</element>
```

Example:

```
<!-- accelerator component q1 is installed at 1.5 m in respect to the beginning of sector -->
<frame at="1.5" ref="q1"/>

<!-- another component d2 is installed at 3.4 m in respect to the beginning of sector -->
<!-- its position is misaligned in the x-direction, and component is rotated around the x-axis -->
<frame at="3.4" ref="d2">
  <offset x="0.001" />
  <rotation phi = "0.002" />
</frame>
```

ADXF Element

Definition:

Element is a leaf component in the accelerator tree organization. There are many different types of accelerator elements (e.g. sbend, quadrupole, etc.) , but all of them have the same structure: name, length, and an open collection of attribute sets. Elements may have a reference to the design element.

XML Schema:

```
<element name = "element">
  <complexType>
    <attribute name="type" />
    <attribute name="name" />
    <attribute name="l" />
    <attribute name="design" />
    <sequence>
      <element name="attributeSet" minOccurs="0" maxOccurs="unbounded" />
    </sequence>
  </complexType>
</element>
```

Example:

```
<!-- design quadrupole q1 -->
<element type="quadrupole" name="q1" l="1.2" >
  <mfield b = "0. 0.024" />
</element>
<!-- real quadrupole q1_1 with a set of measured multipole harmonics -->
<element name="q1_1" design="q1">
  <mfield b = "0. 0.025 0.0004" />
</element>
```

ADXF Element Attribute Set

Definition:

Element Attribute Set is a basic class of different attribute sets associated with specific single physical effects or features (e.g. magnetic field, aperture, etc.)

XML Schema:

```
<complexType name="attributeSetType"/>
<element name="attributeSet" type="attributeSetType"/>

<complexType name="mfieldType">
  <complexContent>
    <extension base="attributeSetType">
      <attribute name="ml" type="expressionType" />
      <attribute name="a" type="arrayType" />
      <attribute name="b" type="arrayType" />
    </extension>
    <complexContent>
      <complexType>
        <element name="mfield" type="mfieldType" substitutionGroup="attributeSet"/>
      </complexType>
    </complexContent>
  </complexContent>
</complexType>
```

Example:

```
<!-- real quadrupole q1_1 with a set of measured multipole harmonics -->
<element name="q1_1" design="q1">
  <mfield b = "0. 0.025 0.0004" />
</element>
```

Examples:

Design description (extract from the RHIC lattice)

```
<adx>
<parameters>
<constant name="alpha" value="0.0" />
<constant name="rhod0" value="236.329952479" />
<constant name="rhodx" value="196.1858025048" />
<constant name="lcend0" value="3.58870392002" />
<constant name="lcendx" value="3.7" />
<constant name="ld0flx" value="0.884076197994" />
<constant name="ld0fla" value="0.1589964" />
<constant name="thd0" value="sin( sin(alpha) + lcendx/ rhodx - lcend0/ rhod0" />
<constant name="thdx" value="sin( sin( alpha ) + lcendx / rhodx)" />
<constant name="ld0" value="rhod0*(thd0-thdx)" />
</parameters>
<elements>
<element type="sbend" name="d0mp08" l = "ld0">
  <bend angle="thd0- thdx" />
</element>
<element type="drift" name="od0flx" l="ld0flx" />
<element type="drift" name="od0fla" l="ld0fla" />
<element type="marker" name="erd0mp08" />
<element type="marker" name="eld0mp08" />
</elements>
<sectors>
<sector name="d008b" line="od0flx, erd0mp08, d0mp08, eld0mp08, od0fla" />
</sectors>
```

Examples:

Operational description (extract from the RHIC lattice)

copy of the design description

```
<adx>
<parameters>
  <constant name="alpha" value="0.0" />
  <constant name="rhod0" value="236.329952479" />
  <constant name="rhodx" value="196.1858025048" />
  <constant name="lcend0" value="3.58870392002" />
  <constant name="lcendx" value="3.7" />
  <constant name="ld0flx" value="0.884076197994" />
  <constant name="ld0fla" value="0.1589964" />
  <constant name="thd0" value="sin( sin(alpha) + lcendx/ rhodx - lcend0/ rhod0) />
  <constant name="thdx" value="sin( sin( alpha ) + lcendx / rhodx)" />
  <constant name="ld0" value="rhod0*(thd0-thdx)" />
</parameters>
<elements>
  <element type="sbend" name="d0mp08" l="ld0">
    <bend angle="thd0-thdx" />
  </element>
  <element type="drift" name="od0flx" l="ld0flx" />
  <element type="drift" name="od0fla" l="ld0fla" />
  <element type="marker" name="erd0mp08" />
  <element type="marker" name="eld0mp08" />
  <element name = "bi8-dh0" design = "d0mp08" >
    <mfield b = "0 0 0.005476 0.033503 -16.11 316.9 1324982 2770273 -10008730294 207554314866"
           a = "0 0 -0.010166 0.024366 19.257 316.9 -318977 -2770273 2859637227" />
  </element>
</elements>
<sectors>
  <sector name="d008b" >
    <frame ref="erd0mp08" at="ld0flx" />
    <frame ref="bi8-dh0" />
    <frame ref="eldmp08" />
    <frame ref="od0fla" />
  </sector>
</sectors>
</adx>
```

Appendix A: ADXF 2.0 vs MAD-X Design Lattice description

MAD-X:

```
LD0 := -RHOD0 * (THD0 - THDX);
```

```
D0MP08: SBEND, L := LD0, ANGLE := THD0 - THDX;
```

```
D008B: LINE = ( OD0FLX, ERD0MP08, D0MP08, ELD0MP08, OD0FLA );
```

ADXF:

```
<constant name="Id0" value = "-RHOD0 * (THD0 - THDX)"/>
<element type="sbend" name="D0MP08" l = "Id0">
    <bend angle="THD0 – THDX" />
</element>
```

```
<sector name="D008B" line="OD0FLX, ERD0MP08, D0MP08, ELD0MP08, OD0FLA" />
```

Two variants

```
<sector name="D008B" >
    <frame ref = "OD0FLX" />
    <frame ref = "ERD0MP08" />
    <frame ref = "D0MP08" />
    <frame ref = "ELD0MP08" />
    <frame ref = "OD0FLA" />
</sector>
```

ADXF 2.0 vs MAD-X (continued):

□ Common features:

- Both formats include **parameters and expressions**. Just like MAD-X, the ADXF public application will support this feature in input files only. But it does not prevent other programs or future versions to preserve parameters in output files as well.
- Both formats support **sharing of design elements** by different sectors.

□ Conceptual differences:

- ADXF 2.0 **does not support automatic name generation**. In ADXF 2.0, each element appearing in a sector must be explicitly defined in the “elements” part. **This approach eases many issues** (e.g. naming, access to elements) **but eliminates some features** (e.g. random error distribution in the design lattice). So far, we don’t have a better solution. Any suggestions are welcome.

Appendix B: ADXF 2.0 vs SXF Operational Lattice Description

SXF:

```
bi8-dh0
sbend {
    tag = d0mp08
    at = 661.74662424
    l = 3.58896623069
    body = { kl = [ -0.0151862520728 ] }
    body.dev = { kl = [ 0 0 0.005476 0.033503 -16.112848 316.932487 1324982.270185 2770273.563708
                    -10008730294.68691 207554314866.7792 ]
                 kls = [ 0 0 -0.010166 0.024366 19.256818 316.932487 -318977.213193 -2770273.563708
                         2859637227.053402 ]
    }
};
```

ADXF:

```
<element type = "sbend" name = "d0mp08" l = "3.58896623069">
    <bend angle= "-0.0151862520728 ">
</element>
<element name = "bi8-dh0" design = "d0mp08" >
    <mfield b = "0 0 0.005476 0.033503 -16.112848 316.932487 1324982.270185 2770273.563708
                -10008730294.68691 207554314866.7792"
            a = "0 0 -0.010166 0.024366 19.256818 316.932487 -318977.213193 -2770273.563708
                2859637227.053402"
    />
</element>
<sector name="blue" >
    ...
        <frame at="661.74662424" ref="bi8-dh0">
    ...
</sector>
```

ADXF 2.0 vs SXF (continued)

□ Common features:

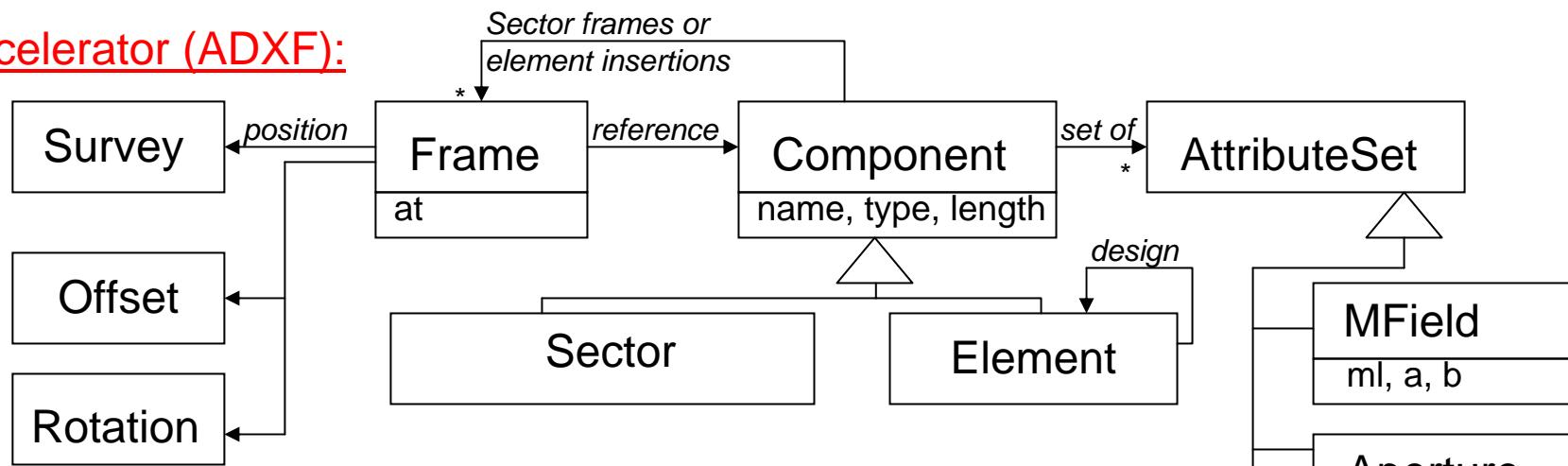
- Both formats are based on the UAL accelerator model
- ADXF is able to accommodate the RHIC SXF description

□ Conceptual differences:

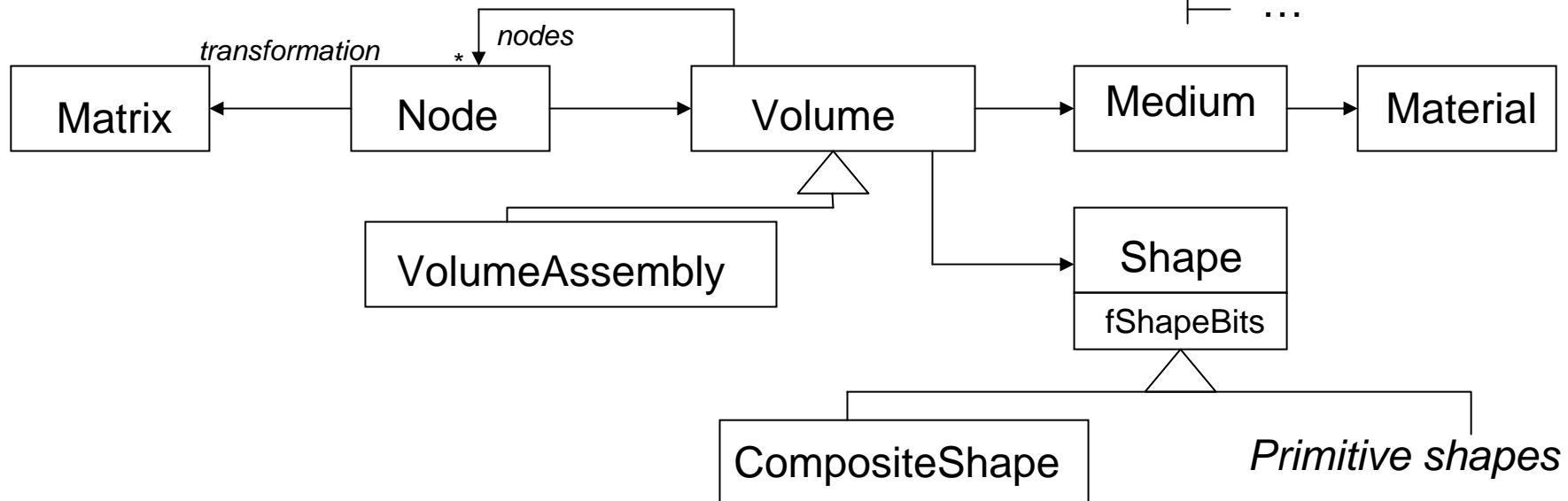
- SXF flat representation is divided into two ADXF parts: elements and sectors; **So, ADXF 2.0 is not flat anymore!** But its structure looks simpler for parser (and for us). Any comments are welcome!
- SXF deviations are replaced with a pair of design and real elements;
- new extensions

Appendix C: ADXF Accelerator vs ROOT Detector Models

Accelerator (ADXF):



Detector (ROOT):



ADXF Accelerator vs ROOT Detector Models (continued)

| ADXF Concept | ADXF Accelerator Model | ROOT Detector Model |
|----------------------------------|---|--|
| Accelerator | Accelerator: any selected sector | Detector: any selected volume |
| Sector | Sector: sequence of frames with installed components. Sector does not have its own attribute sets | VolumeAssembly: container of nodes (positioned volumes); it does not have its own attributes, such as shape and medium |
| Frame | Frame: an layout of an installed accelerator component. It has a relative position, misalignments, and a reference to a component (sector or element) | Node: positioned volume. It has a relative position (geometrical transformation) and a reference to a volume. |
| Position | Longitudinal position “at” or its Survey coordinates in the sector local coordinate system. | Matrix: geometrical transformation from parent to local reference frame. |
| Element | Element: is a fully defined accelerator object having a type, collection of attribute sets, and possibly containing a list of node-insertions. | Volume: fully defined geometrical object having a given shape, medium, and possibly containing a list of nodes. |
| Element Type | Component’s attribute used by the corresponding tracker. There are several element types, such as sbend, quadrupole | There is no special volume attribute. But for the sake of comparison, it could be associated with a Shape EShapeType. There are several shape types (primitives) |
| Type-specific sets of attributes | AttributeSet’s of magnetic and RF fields, e.g. MField, Bend, RfField, etc. | Primitive Shape’s |
| Common sets of attributes | Aperture | Material |